HW#00: MLP function approximation

CE397 and CSE393: Scientific Machine Learning

---

1 Theoretical Foundations [2 pts]

1.1 The Universal Approximation Theorem (UAT)

a) In 2-3 lines, state the Universal Approximation Theorem for a single-hidden-layer feedforward neural network.

b) Briefly discuss the limitations of the UAT. What does it guarantee, and what does it *not* guarantee about network training, architecture, or efficiency?

1.2 Backpropagation for a Two-Layer MLP

Consider a two-layer MLP with a single input neuron (x), a hidden layer with H neurons, and a single linear output neuron ($\hat{y}$). The hidden layer uses a sigmoid activation function, $\sigma(z) = (1 + e^{-z})^{-1}$. The network function is:

$$\hat{y}(x;\theta) = \mathbf{w}^{(2)T}\sigma(\mathbf{w}^{(1)}x + \mathbf{b}^{(1)}) + b^{(2)}$$

where $\mathbf{w}^{(1)}, \mathbf{b}^{(1)} \in \mathbb{R}^H$, $\mathbf{w}^{(2)} \in \mathbb{R}^H$, and $b^{(2)} \in \mathbb{R}$. The loss for a single training example (x_j, y_j) is the squared error $L = (\hat{y}(x_j) - y_j)^2$.

Using the chain rule, derive the update rules for the parameters by finding the partial derivatives $\frac{\partial L}{\partial w_i^{(2)}}$, $\frac{\partial L}{\partial b^{(2)}}$, $\frac{\partial L}{\partial w_i^{(1)}}$, and $\frac{\partial L}{\partial b_i^{(1)}}$. Recall that $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.

2 Computational Experiments: Function Approximation [4 pts]

2.1 Approximating a Discontinuous Function

This problem explores the challenge of approximating non-smooth functions. Consider the Heaviside step function on the interval $x \in [0, 1]$:

$$H(x) = \begin{cases} 0 & x < 0.5 \\ 1 & x \geq 0.5 \end{cases}$$

a) Using a single hidden layer MLP with 50 neurons and a 'tanh' activation, train the network to approximate $H(x)$.

b) Plot the learned function $\hat{u}(x)$ against the true function $H(x)$. Describe the behavior around the discontinuity at $x = 0.5$. Does it resemble the Gibbs phenomenon?

c) Experiment to improve the approximation by independently:

- Increasing the hidden layer width (e.g., to 200, then 500 neurons).
- Increasing the network depth (e.g., 3 hidden layers of 50 neurons each).
- Changing the activation function to ReLU.
- Try different loss functions

d) **Analysis:** Why does a 'tanh'-based network struggle with discontinuities? Which modification produced the best result and why? Relate your findings to the fact that compositions of 'tanh' are C^∞, while ReLU networks are piecewise linear.

2.2 Comparing Activation Functions & Optimizers

For the smooth function $u(x) = 0.5\cos(\pi x^2)$:

a) Compare the performance of the following activation functions: Sigmoid, Tanh, ReLU, Leaky ReLU, and SiLU (Swish). For each, plot the final learned function and the training loss vs. epochs on a single log-log plot.

b) Using the best-performing activation function from part (a), compare the convergence behavior of the following optimizers: SGD (with momentum=0.9), RMSprop, and Adam. Plot their training loss curves on a single log-log plot and discuss the differences in convergence speed and stability.

3 Compositional Learning [4 pts]

This section explores how to solve complex problems by breaking them into simpler, modular components that can be learned separately and then combined.

3.1 Learning and Composing Multi-Scale Functions

Consider the function $h(x) = f_{\text{linear}}(x) + f_{\text{high}}(x)$ on the interval $x \in [0, 1]$, where:

- $f_{\text{linear}}(x) = 1 + 2x$ (a simple linear trend)
- $f_{\text{high}}(x) = 0.5\cos(20\pi x)$ (a high-frequency component)

Part 1: Learning the High-Frequency Component

To understand the challenge, first focus solely on approximating $f_{\text{high}}(x)$.

a) Train MLPs of increasing depth (1, 2, 3, and 4 hidden layers) to learn $f_{\text{high}}(x)$.

b) Plot the results on a 2x2 grid to visually demonstrate how approximation quality improves with depth. Explain this result in the context of spectral bias.

Part 2: Composition by "Stitching" Networks

Now, we will tackle the full problem $h(x)$ using a "divide and conquer" strategy.

a) Modular Training:

- Train a simple, shallow network, N_{linear}, to learn $f_{\text{linear}}(x)$.
- Train a separate, deep network, N_{high}, to learn $f_{\text{high}}(x)$.

b) Manual Composition: The final stitched model is formed by adding the outputs of the two trained networks:

$$\hat{h}_{\text{stitched}}(x) = N_{\text{linear}}(x) + N_{\text{high}}(x)$$

Implement this composition in code.

c) Comparison & Analysis: Train a single, deep end-to-end network to learn $h(x)$ directly. Create a side-by-side plot comparing the result of your stitched model against the end-to-end model. For clarity, zoom in on the interval $x \in [0.2, 0.8]$ for one of your plots. Discuss the results. Which approach is more effective and why?